Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Interval-based Recording of Generated Pseudorandom Numbers

Violeta Tomašević[1], Milo Tomašević[23], Slobodan Bojanic[4]

1 Singidunum University, Danijelova 32, 11010 Belgrade, Serbia

2 School of Electrical Engineering, University of Belgrade, Bul. kralja Aleksandra 73,

3 11120 Belgrade, Serbia

4 Universidad Politécnica de Madrid

Abstract

For some applications that use pseudorandom numbers it is essential to keep the record of numbers generated so far. Such a representative example is cryptanalytic TMTO approach. In order to save the space, instead of straightforward recording of individual numbers generated, an ordered tree-like data structure which tracks the intervals of generated numbers is proposed. For estimating the memory requirements of this structure as a function of pseudorandom numbers range size, an analytical probabilistic model is established and used. This model determines the maximum number of intervals during recording which corresponds to the tree size. The result obtained from analytical model is fully validated experimentally by means of simulation for a wide spectrum of range sizes.

OPEN ACCESS

Published: 25/06/2019

Accepted: 11/06/2019

Submitted: 07/06/2019

DOI: 10.23967/j.rimni.2019.06.003

Keywords:
'"pseudorandom numbers
interval trees
simulation analysis

1. Introduction

Random events are immanent to numerous natural phenomena. Being fully unpredictable they can be modelled by random number sequences which do not follow any specific rules. Computer simulation of random processes requires computer-based generation of random numbers [1]. However, in practice such computer generated number sequences are not entirely random since detereministic algorithms are employed for this purpose. It means that for the same initial state these algorithms always produce the same sequences of numbers. This is the reason why the computer generated numbers are regarded as pseudorandom. Although true randomness can not be expected to achieve in a computer environment, quite often a sequence of pseudorandom numbers satisfies the most of the statistical testes of randomness [2-6]. Without a precise definition of relevant tests which would guarantee the reliability of obtained results, the implementation of pseudorandom generators is usually based on detailed mathematical analysis of their characteristics.

Pseudorandom numbers have a widespread application in various fields. The examples of their intensive use are the areas of statistics, gambling, probability and chance games, computer simulations, cryptography, etc. In some aplications, it is quite essential to constantly keep the record of already generated pseudorandom numbers.

An example that represents an immediate motivation for the analysis conducted in this paper is found in the area of cryptanalysis. This is a well-known Hellman TMTO algorithm which solves the main cryptanalytical problem of inverting one-way function in order to find the secret key [7]. This method, as well as its follow-up improvements [8,9], tries to achive a trade-off between time and memory requirements in this very computationally demanding task. An off-line, precomputation phase of the algorithm generates the fixed-length chains of keys (which can be regarded as pseudorandom), so that an on-line, attack phase can perform more efficient search. However, the main problem is that no record is kept during generation of chains about keys already included, so the multiple occurence of some keys in chains is inevitable. This leads to some irregular situations like looping and merging of chains. On the other hand, since the total number of keys in chains is equal to the size of the key space, multiple occurence of some keys prevents some other keys to be included into chains which incures a lower coverage of the key space. The consequence is that such keys can not be found at all, while the repeated key values are responsible for erroneous situations (false alarms). Therefore, the success of the attack is impaired, making the method probabilistic. The problems can be avoided by keeping the record of the randomly chosen keys which are already included into chains during off-line phase. In this way, resulted *perfect chains* avoid key repetion and attain full key space coverage, making the method deterministic. The relevant characteristics of perfect chains are examined in [10] by using probabilitic analytical and simulation model.

Pseudorandom number generator produces the numbers by a random choice from a given set, usually a range of numbers. If the range of possible numbers is quite large, which is an often case, the tracking of already generated numbers can be practically infeasible because of the extreme space demands. In order to decrease the memory required for tracking, instead of

Correspondence: Violeta Tomašević (vitomasevic@singidunum.ac.rs), Milo Tomašević (mvt@etf.rs), Slobodan Bojanic (slobodan.bojanic@upm.es).

keeping each individual generated number, only the record of intervals of generated numbers can be kept. Consequently, an ordered dynamic tree-like structure is proposed where each node represents an interval of generated numbers with its lower and upper bounds. As the new numbers are generated and the tree updated accordingly, it is expected that number of nodes increases from one to some maximum number. Then, as new numbers are generated, the nodes are expeted to coalesce and the number of nodes decreases until all numbers are generated and the tree colapses to one node. The main goal of this paper is to determine the maximum number of nodes as a function of size of the generated numbers range.

A precise formal procedure of keeping the record during random number generation and the problem statement are given in Section 2. Then, problem is theoretically solved by an analytical approach which is described in Section 3. The result of this analysis is validated by means of simulation evaluation presented in Section 4. Finally, the conclusion is drawn in Section 5.

2. Problem statement

Let the random number generator (RNG) generates integers from the given range [1..*N*] (denoted as set *R*) long enough to guarantee that each number from *R* appears at least once. After RNG generates a number, the check is made if this number is already generated before. If not, it is used in application and some record of numbers generated so far is updated; otherwise, RNG generates a new number. The process is carried on until all numbers from *R* are generated and included into the record.

For the sake of clarity of the following presentation and analysis, during the process of random generation we will maintain two sets. The record of numbers generated so far is kept as a set of integers (denoted as *I*). In order to save the space, it is implemented in an appropriate form of intervals of consecutive integers. These intervals of already used numbers will be referred as *I*–intervals. The remaining numbers from *R* are kept in a set of unused integers (denoted as *S*), also implemented in the form of intervals (*S*–intervals). In both set implementations, only lower and upper bound for each interval are recorded in memory. The interval can have only one element, referred to as *single element interval* (e.g., [*a*, *a*]), or more than one element, referred to as *multiple element interval* (e.g., [*a*, *b*], *a* < *b*). Since *S* and *I* sets are complements in respect to set *R* (i.e., *S* + *I* = *R*), in practical implementation it is sufficient to maintain only one of these sets for memory efficiency. Figure 1 illustrates alternating layout of *I*–intervals and *S*–intervals within the set *R*. Used numbers are denoted as circles and *I*–intervals are represented as shaded boxes, while unused numbers are denoted with squares and *S*–intervals are represented as white boxes.

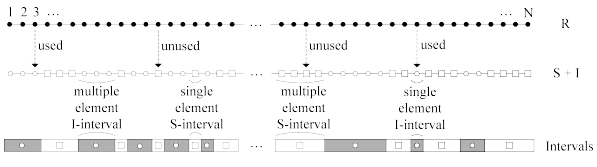

Figure 1. Layout of *I*–intervals and *S*–intervals

Let we start with set of integers $S = \{x \mid 1 \le x \le N\}$ (i.e., $S = R$) and an empty set $I = \emptyset$. In each step, we randomly choose an element x_i from *S* and move it to *I*; i.e., $S = S - [x_i]$ and $I = I + [x_i]$. Consequently, after *N* steps we will have $S = \emptyset$ and $I = R$. In each step, the record of numbers already chosen and included into *I* is updated in the following way:

A. In step $t = 1$, an element $x_1 \in S$ is randomly chosen and included into *I*. Now, there is only one *I*-interval, $[x_1,x_1]$.

B. In step $t = i$, $2 \le i \le N$, currently randomly chosen element $x_i \in S$ is checked for adjacency to existing *I*–intervals with three possible outcomes:

1. if it is not adjacent to any of existing *I*–intervals, a new *I* –interval $[x_i, x_i]$ is created,

2. if it is adjacent to only one interval $[a, b]$, this *I*–interval is extended with x_i
(e.g., if $x_i = a - 1$, interval $[a, b]$ is extended to $[x_i, b]$
or if $x_i = b + 1$, interval $[a, b]$ is extended to $[a, x_i]$),

3. if it is adjacent to two *I*–intervals $[a, b]$ and $[c, d]$ in a way that $x_i = b + 1$ and $x_i = c - 1$, these two intervals are merged into interval $[a, d]$ (x_i previously corresponded to a single element *S*–interval).

Obviously, after an element x_i is included into *I*, three outcomes are possible in a step:

- number of *I*–intervals is increased by 1 (cases A and B.1),
- number of *I*–intervals stays the same (case B.2),
- number of *I*–intervals is decreased by 1 (case B.3).

In step $t = 1$, the number of *I*–intervals is 1. Then, it increases until some maximum *M* is reached as new *I*–intervals are created. After reaching maximum, it is expected that *I*–intervals are progessively coalescing, and their number will fall until only one interval, [1, *N*], remains in step $t = N$ (all elements from *S* are now moved to *I*, $S = \emptyset$ and $I = R$).

For practical viability of this procedure, one of the crucial things is the memory needed for keeping the record of generated elements in set *I* (and also of unused elements in *S*). Memory requirements are directly proportional to *M* since *I* can be implemented as a dynamic structure in which only upper and lower bound of each interval are recorded, as it will be explained in Section 4. In the following sections, the maximum number of *I*–intervals *M* is determined by both analytical and simulation means.

3. Analytical solution

Let us assume that in step $t = i$, $1 \le i \le N$, an element x_i is chosen from *S* and moved to *I*. After that, the number of elements in *S* is $N - i$. Let the number of *I*–intervals after adding x_i to *I* be denoted as r_i. In the next step $t = i + 1$, $i \ne N$, an element x_{i+1} is chosen from *S*. As previosly elaborated, its inclusion into *I* can lead to:

- decreased number of *I*–intervals - $r_i - 1$ (merging of two intervals),
- increased number of *I*–intervals - $r_i + 1$ (creating a new interval),
- the same number of *I*–intervals - r_i (extending an existing interval).

Let the symbols v_g, v_l and v_e stand for probabilities that the number of *I*–intervals will increase, decrease or stay the same, respectively. Also, the following equation must hold

$$v_g + v_l + v_e = 1$$

In a given step, these probabilities depend on current positions of *I*–intervals of used numbers within range *R*. Three possible layouts of range *R* are shown in Figure 2. *I*–intervals are sequenced from 1 to r_i and again represented by shaded boxes, while intervening *S*–intervals of unused numbers are represented by white boxes.

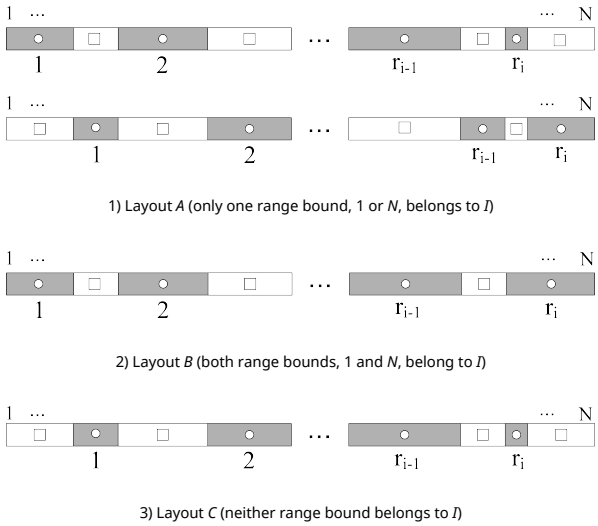

Figure 2. Possible layouts of *I*–intervals within range *R*

Let $v_A(i)$, $v_B(i)$ and $v_C(i)$ denote the probabilities of layouts *A*, *B* and *C* after step $t = i$, respectively, and C_i^N is the number of i -combinations from set of N elements. Then, these probabilities can be calculated as:

$$v_A(i) = 2\frac{C_{i-1}^{N-2}}{C_i^N} = 2\frac{\binom{N-2}{i-1}}{\binom{N}{i}} = 2i\frac{N-i}{N(N-1)}$$

$$v_B(i) = \frac{C_{i-2}^{N-2}}{C_i^N} = \frac{\binom{N-2}{i-2}}{\binom{N}{i}} = i\frac{i-1}{N(N-1)}$$

$$v_C(i) = \frac{C_i^{N-2}}{C_i^N} = \frac{\binom{N-2}{i}}{\binom{N}{i}} = \frac{(N-i)(N-i-1)}{N(N-1)}$$

We will now derive the expressions for probabilities v_l, v_g and v_e in case of each particular layout. To this end, we define an important parameter denoted as m_i. It represents an average number of single element *S*–intervals after step $t = i$. This type of interval is important since its only element which belongs to *S* separates two consecutive *I*–intervals and its selection from *S* and inclusion into *I* in some later step makes that these two *I* –intervals are merged into one.

Let us assume that in step $t = i + 1$ one of $N - i$ elements from *S* is chosen.

Layout A

In case of this layout, the number of *I*–intervals (r_i) is equal to the number of *S*–intervals (Figure 2a). The number of *I*–intervals can be decreased only if the chosen number belongs to some single element *S*–interval other than [1, 1] and [*N, N*]. The probability that single element *S*–interval lies at 1 or N is m_i/r_i, while the probability of the opposite case is $1 - m_i/r_i$. Therefore, the probability of merging two consecutive *I*–intervals is

$$v_l(A) = \frac{m_i}{r_i}\frac{m_i - 1}{N - i} + \frac{r_i - m_i}{r_i}\frac{m_i}{N - i} = \frac{m_i}{N - i}\frac{r_i - 1}{r_i} \quad (1)$$

The number of *I*–intervals can be increased only if the chosen number does not belong to some single element *S*–interval and also if it is not adjacent to some bound of any *I*–interval. The probability of such a case is

$$v_g(A) = \frac{m_i}{r_i}\frac{N - i - m_i - 2(r_i - m_i)}{N - i} + \frac{r_i - m_i}{r_i}\frac{N - i - m_i - 2(r_i - m_i - 1) - 1}{N - i}$$

$$v_g(A) = 1 + \frac{m_i}{N - i}\frac{r_i - 1}{r_i} + \frac{1 - 2r_i}{N - i} \quad (2)$$

The number of *I*–intervals stays the same if the chosen number is adjacent to a bound of some *I*–interval. It can belong to some multiple element *S*–interval or to single element *S*–interval [1, 1] or [*N, N*]. The probability for such a case is

$$v_e(A) = \frac{m_i}{r_i}\frac{2(r_i - m_i) + 1}{N - i} + \frac{r_i - m_i}{r_i}\frac{2(r_i - m_i - 1) + 1}{N - i} = \frac{2m_i(1 - r_i)}{N - i} + \frac{r_i(2r_i - 1)}{N - i} \quad (3)$$

Using expressions (1) – (3), assuming layout *A* after step $t = i$ an average number of *I*–intervals after step $t = i + 1$ can be calculated as

$$r_{i+1}(A) = v_l(A)(r_i - 1) + v_e(A)r_i + v_g(A)(r_i + 1) = \frac{N - i + 1}{N - i} + \frac{N - i - 2}{N - i}r_i \quad (4)$$

Layout B

For this layout, the number of *I*–intervals is r_i while the number of *S*–intervals is $r_i - 1$ (Figure 2b). The number of *I*–intervals will be decreased only if the chosen number corresponds to some single element *S*–interval. The probability for such an event when two *I*–intervals are merged is

$$v_l(B) = \frac{m_i}{N - i} \quad (5)$$

The number of *I*–intervals will be increased if the chosen number neither corresponds to any single element *S*–interval nor is adjacent to a bound of any *I*–interval. Then, a new *I* –interval is created with probability

$$v_g(B) = \frac{N - i - m_i - 2(r_i - 1 - m_i)}{N - i} \quad (6)$$

The number of *I*–intervals does not change if chosen number corresponds to a bound of some multiple element *S*–interval. It can happen with probability

$$v_e(B) = \frac{2(r_i - 1 - m_i)}{N - i} \quad (7)$$

Based on expressions (5) – (7), assuming layout *B* after step $t = i$ an average number of *I*–intervals after step $t = i + 1$ can be calculated as

$$r_{i+1}(B) = v_l(B)(r_i - 1) + v_e(B)r_i + v_g(B)(r_i + 1) = \frac{N - i + 2}{N - i} + \frac{N - i - 2}{N - i} r_i \quad (8)$$

Layout C

For this layout, the number of *I*–intervals is r_i while the number of *S*–intervals is $r_i + 1$ (Figure 2c). The number of *I*–intervals will be decreased if the chosen number corresponds to some single element *S*–interval other than [1, 1] and [*N, N*].

The probability v_0 that no one single element *S*–interval lies on range bounds 1 and *N* is

$$v_0 = \frac{\binom{r_i - 1}{m_i}}{\binom{r_i + 1}{m_i}} = (r_i + 1 - m_i)\frac{r_i - m_i}{r_i(r_i + 1)}$$

The probability v_1 that one single element *S*–interval lies on a range bound (1 or *N*) is

$$v_1 = 2\frac{\binom{r_i - 1}{m_i - 1}}{\binom{r_i + 1}{m_i}} = 2m_i\frac{r_i + 1 - m_i}{r_i(r_i + 1)}$$

The probability v_2 that two single element *S*–intervals lie on range bounds 1 and *N* is

$$v_2 = \frac{\binom{r_i - 1}{m_i - 2}}{\binom{r_i + 1}{m_i}} = m_i\frac{m_i - 1}{r_i(r_i + 1)}$$

Then, the probability that number of *I*–intervals will be decreased is

$$v_l(C) = v_2\frac{m_i - 2}{N - i} + v_1\frac{m_i - 1}{N - i} + v_0\frac{m_i}{N - i} = \frac{m_i}{N - i}(v_2 + v_1 + v_0) - \frac{2v_2 + v_1}{N - i} \quad (9)$$

The number of *I*–intervals will be increased if the chosen number neither corresponds to any single element *S*–interval nor is adjacent to a bound of any *I*–interval. The probability of such a case is:

$$v_g(C) = v_2\frac{N - i - m_i - 2(r_i + 1 - m_i)}{N - i} + v_1\frac{N - i - m_i - 2(r_i - m_i) - 1}{N - i} + v_0\frac{N - i - m_i - 2(r_i - m_i)}{N - i}$$
$$v_g(C) = \frac{N - i - m_i - 2(r_i - m_i)}{N - i}(v_2 + v_1 + v_0) - \frac{2v_2 + v_1}{N - i} \quad (10)$$

The number of *I*–intervals stays the same if the chosen number corresponds to a bound of some multiple element *S*–interval or to a single element *S*–interval at any bound of range *R* (1 or *N*). It can happen with probability

$$v_e(C) = v_2\frac{2 + 2(r_i + 1 - m_i)}{N - i} + v_1\frac{2 + 2(r_i + 1 - m_i - 1)}{N - i} + v_0\frac{2 + 2(r_i + 1 - m_i - 2)}{N - i}$$
$$v_e(C) = \frac{2(r_i - m_i)}{N - i}(v_2 + v_1 + v_0) + 2\frac{2v_2 + v_1}{N - i} \quad (11)$$

Since it can be calculated from above that

$$\frac{2v_2 + v_1}{N - i} = \frac{2m_i}{(r_i + 1)(N - i)}$$

and it holds that

$$v_0 + v_1 + v_2 = 1,$$

the expressions (9) – (11) become:

$$v_l(C) = \frac{m_i}{N - i}\frac{r_i - 1}{r_i + 1} \quad (12)$$

$$v_g(C) = \frac{1}{N - i}(N - i + m_i - 2r_i - \frac{2m_i}{r_i + 1}) \quad (13)$$

$$v_e(C) = \frac{2}{N - i}(r_i - m_i + \frac{2m_i}{r_i + 1}) \quad (14)$$

From expressions (12) – (14), assuming layout *C* after step $t = i$, the average number of *I*–intervals after step $t = i + 1$ can be derived as:

$$r_{i+1}(C) = v_l(C)(r_i - 1) + v_e(C)r_i + v_g(C)(r_i + 1) = 1 + \frac{N - i - 2}{N - i} r_i \quad (15)$$

Expressions (4), (8) and (15) shows that an average number of *I* –intervals (r_{i+1}) does not depend on the number of single element *S*–intervals (m_i) for all three possible layouts *A*, *B* and *C*.

Finally, an average number of intervals r_{i+1} after step $t = i + 1$ can be calculated as:

$$r_{i+1} = v_A(i)r_{i+1}(A) + v_B(i)r_{i+1}(B) + v_C(i)r_{i+1}(C)$$
$$r_{i+1} = 1 + \frac{2i}{N(N - i)} + (1 - \frac{2}{N - i})r_i \qquad 1 \le i \le N - 1,$$
$$r_1 = 1 \quad (16)$$

Solving the recurrence from (16), we obtain:

$$r_{i+1} = (i + 1)\frac{N - i}{N} \quad (17)$$

Let $f(i) = r_{i+1}$. The average number of *I*–intervals as a function of current number of steps *i* is shown in Figure 3. First and second derivation of this function are

$$f'(i) = \frac{N - 1}{N} - \frac{2i}{N} \quad \text{and} \quad f''(i) = -\frac{2}{N},$$

respectively. Since $f''(i) < 0$, $f(i)$ has a local maximum for $i_0 = \frac{N - 1}{2}$ ($f'(i_0) = 0$). The value of this local maximum is

$$M = f(i_0) = \frac{(N + 1)^2}{4N}.$$

For $N >> 1$ it gives $M \approx N/4$, which represents the solution of the stated problem.

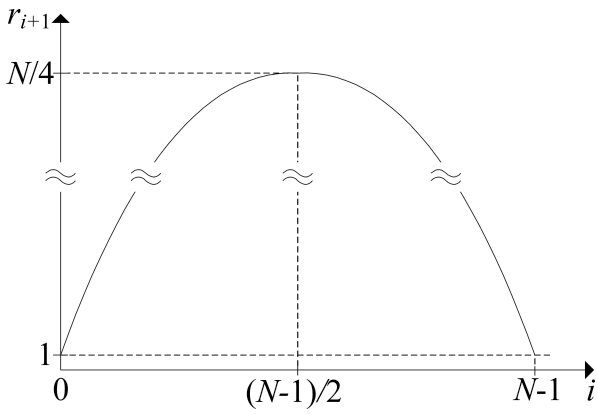

Figure 3. An average number of I–intervals

4. Simulation results

Theoretical finding of the analytical approach conducted in the previuos section is also verified by means of software simulation. An appropriate program that simulates the procedure of random number generation elaborated in Section 2 is implemented. Random number generator from [11] is exploited in this simulation. For keeping the record of randomly numbers generated so far (implementation of set I) a structure called *interval tree* is used. It is a modification of standard binary search tree [12], whose nodes correspond to I –intervals instead of single values. An example of such a tree is given in Figure 4.

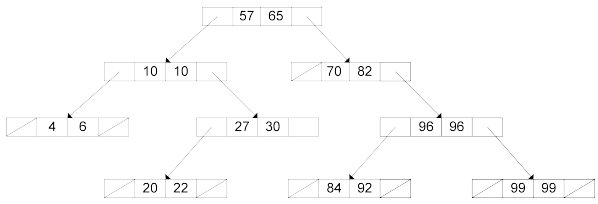

Figure 4. An example of an interval tree

Each node has two integer fields for lower and upper bounds of the corresponding interval and two pointer fields to its left and right subtrees, so memory consumed is $O(M)$ where M is the maximum number of nodes (I –intervals). All intervals in the left subtree are lower, while the intervals in the right subtree are higher, so an efficient binary search is possible with average time complexity of $O(\log M)$ [12].

The experimental results are presented in Table 1. The size of random number range (N – first column) is varied from 100 to 10000000. For each value of N, the results are averaged over 10 experiments. Theoretically expected number of I – intervals is calculated from analytical model as $M_a = \frac{(N+1)^2}{4N}$ and given in the second column. The statistics about the maximum number of nodes (I – intervals) in the interval tree is collected (M_s – third column). Fourth column shows a relative difference between the results of analytical and simulation models calculated as $d_m = abs((M_a - M_s)/M_a)$. As this difference decreases with N and become negligible, it is evident that the results of analytical model perfectly match the experimental results. Also, the number of generated values included into tree when the number of nodes reaches the maximum M is also kept (P – fifth column). Then, an average size of an I – interval is calculated as $l_{avg} = P/M_s$ (sixth column).

Table 1. Experimental results

N	M_a	M_s	d_m (%)	P	l_{avg}
100	25.5025	27	0.058720	47	1.74
1000	250.50025	253	0.009979	482	1.90
10000	2500.500025	2509	0.003399	4945	1.97
100000	25000.5000025	25020	0.000780	50060	2.00
1000000	250000.50000025	250015	0.000058	500278	2.00
10000000	2500000.500000025	2494898	0.000041	5005260	2.00

5. Conclusion

Random number generation is widely exploited in many applications. Not rarely it is required that each number from the given range should be chosen and used in application only once. In that case it is necessary to keep the record of already generated numbers. For very large range sizes the recording procedure can be extremely space demanding. The goal of this paper is to estimate the memory needed for this purpose.

Instead of straightforward recording of each individual number, an advanced data structure in form of an interval tree is employed. It allows for an efficient binary search, and each node represents an interval of already generated numbers keeping only upper and lower bound for each interval. Memory demands are directly proportional to maximum number of nodes in this tree. These demands are first estimated using an analytical probabilistic model. It was obtained that for a range size N the maximum number of intervals (or nodes in the interval tree) is $N/4$ approximately. This finding is proved by simulation means since the experimental results entirely verified the theoretical one. The fact that the memory savings are within a constant factor of linear memory complexity implies that such a recording is still practically infeasible for very large range sizes.

Acknowledgement

This work has been partially supported by the Serbian Ministry of Education and Science (the projects III 44006 and III 44009).

References

[1] Knuth, D. The Art of computer programming, 3rd ed. Vol. 2: Seminumerical Algorithms, Addison Wesley, 1997.

[2] Zaman S.U., Ghosh R. Review on fifteen statistical tests proposed by NIST. J. of Theor. Phys. and Cryptogr., 1:18-31, 2012.

[3] Senn S.J., Rothman K.J., Carlin J.B., Poole C., Goodman S.N., Altman D.G. Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations. Eur. J. of Epidemiol., 31:337-350, 2016.

[4] Doganaksoy A., Sulak F., Uguz M., Seker O., Akcengiz Y. New statistical ramdomness tests based on length of runs. Math. Probl. in Eng., 2015:14 pages, 2015.

[5] Mohammad O.J., Abbas S., Horbaty E., Salem A. A new trend of pseudo random number generation using QKD. Int. J. of Comput. Appl., 96:13-17, 2014.

[6] Liu C., Lai X. Evaluation of statistical tests for randomness using conditional entropy. Int. Conf. on Cyber-enabled Distrib. Comput. and Knowl. Discov., 504-509, 2013.

[7] Hellman M. A cryptanalytic time-memory trade-off. IEEE Trans. on Inf. Theory, 4:401-406, 1980.

[8] Standaert F., Rouvroy G., Quisquater J., Legat J. A time memory tradeoff using distinguished points; new analysis & FPGA results. Lect. Notes in Comput. Sci., 2523:593-609, 2003.

[9] Oechslin P. Making a faster cryptanalytic time-memory trade-off. Lect. Notes in Comput. Sci., 2729:617-630, 2003.

[10] Tomašević V., Tomašević M. An analysis of chain characteristics in the cryptanalytic TMTO method. Theor. Comput. Sci., 501:52-61, 2013.

[11] Matsumoto M., Nishimura T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. on Model. and Comput. Simul. 8:3-30, 1998.

[12] Cormen, T., Leiserson, C., Rivest, R., Stein, C. Introduction to algorithms. Third ed. MIT Press, 2009.